Reasoning about Real-Time Systems with Temporal Interval Logic Constraints on Multi-State Automata*

Armen Gabrielian

Thomson-CSF, Inc.
630 Hansen Way, Suite 250
Palo Alto, CA 94304

Abstract

Models of real-time systems using a single paradigm often turn out to be inadequate, whether the paradigm is based on states, rules, event sequences or logic. In this paper, a model-based approach to reasoning about real-time systems is presented in which a temporal interval logic called TIL is employed to define constraints on a new type of high-level automata. The combination, called "Hierarchical Multi-State (HMS) machines," can be used to model formally a real-time system, a dynamic set of requirements, the environment, heuristic knowledge about planning-related problem solving, and the computational states of the reasoning mechanisms. In this framework, mathematical techniques have been developed for (1) proving the *correctness* of a representation, (2) *planning* of concurrent tasks to achieve goals, and (3) *scheduling* of plans to satisfy complex temporal constraints. HMS machines allow reasoning about a real-time system from a model of *how* truth arises instead of merely depending on *what* is true in a system.

1. Introduction

Real-time systems are characterized by unpredictability of inputs and "hard deadline" requirements. In addition, since many real-time systems are utilized in life-critical situations, strict "safety properties" are usually defined for them. A safety property is a state of affairs that must always remain true in a system. Instead of the usual discussion of "liveness properties," it is useful to define other requirements of a real-time system in terms of a set of "conditional goals" defined in terms of (condition, goal) pairs. A condition defines the state of affairs under which the associated goal must be pursued. We assume that deadlines may be associated with goals and that requirements are dynamic so that the pursuit of an active goal may have to be abandoned if certain other conditions become true. Thus, at the specification stage, the main forms of reasoning about a real-time system consists of the *verification* that (1) safety properties are not violated and (2) conditional goals are achievable. For traditional systems which operate deterministically or stochastically, this is essentially sufficient even though it can be a very complicated process. At the operational stage, two other forms of reasoning arise for "intelligent systems" which are not defined deterministically and require a search or other forms of analysis to instantiate a specific set of responses in a particular situation. First, off-line reasoning can be performed to determine in advance a set of allowable actions to achieve goals. Secondly, on-line reasoning can be employed, where deadlines on the reasoning process itself may have been defined. A key problem in the specification and operation of complex real-time systems is the choice of a representational framework that can provide manageable approaches to specification, verification, and instantiation of behavior.

While numerous formal representational schemes have been proposed for systems in general and real-time systems in particular, most of these are based on one of the following paradigms: state-based

* The work reported in this paper was supported in part by the Office of Naval Research under Contract N00014-89-C-0022.

models, rules, event sequences or logic. Two major examples of state-based models are automata and Petri nets. For real-time systems, traditional automata are inadequate for at least two important reasons: (1) explosion of the state space for non-trivial systems, and (2) absence of a natural mechanism for representing temporal constraints. Petri nets reduce the state space and can represent concurrent activities adequately. However, in Petri nets numerous dummy states are usually necessary to maintain logical consistency and no clear separation is made between *precedence* and *causality* [7]. In addition, even timed Petri nets have a limited language for representing temporal relationships among states and events [6]. Specification and verification of complex real-time requirements are also difficult for rule-based systems and event sequences. In particular, it is generally accepted that while rules are appropriate for defining prototypical behavior, they are inadequate for reasoning about novel situations. As far as pure logical formalisms are concerned, temporal logic provides a promising approach, except for two shortcomings. First, certain simple regular properties cannot be expressed in temporal logic [10]. Secondly, in a pure logic-based language a system is represented merely in terms of *what* is true. This gives a limited understanding of system behavior, since knowledge of *how* truth arises which is common to state models is not readily available.

The purpose of this paper is to present a brief overview of a comprehensive framework for specifying real-time systems and reasoning about them, called "Hierarchical Multi-State (HMS) machines," that integrates high-level "multi-state" automata and fragments of a temporal interval logic called TIL ([7], [4], [3], [5], [6]). As noted in Figure 1, an HMS machine can be used to define formally the dynamic behavior of a system, its requirements, a model of the environment, heuristic knowledge about planning-related problem solving, and the state of the computational resources used in reasoning. Given such a specification, the system can be simulated, its correctness can be verified formally, and it can be used for both off-line and on-line reasoning to derive operational plans and schedules to respond to the dynamics of a real-time situation.

Section 2 presents an outline of a simple form of HMS machines, with a brief discussion of the method for representing requirements in terms of "policy HMS machines." Section 3 presents an overview of the planning process, plan representation languages and a scheduling algorithm for plans. Section 4 presents a brief set of conclusions and directions for future work.

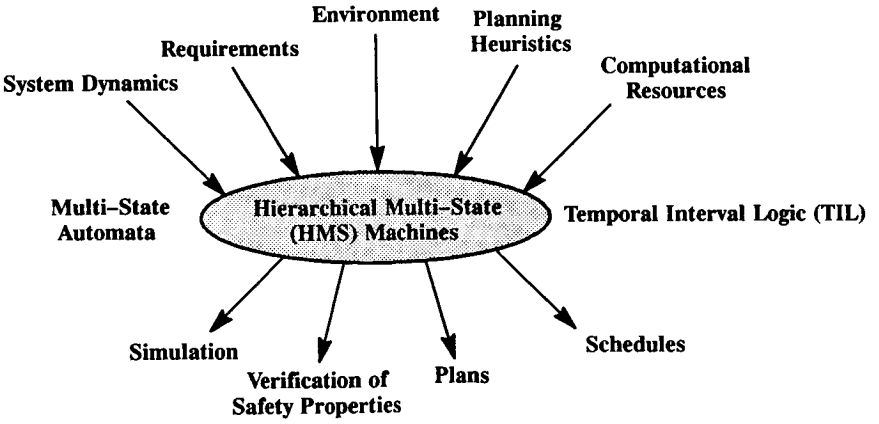

Figure 1. Specification, Verification and Reasoning Framework for Real-Time Systems

2. Automata, Temporal Logic, Machines and Real-Time Systems

An automaton consists of a set of "states" and a set of "transitions" that cause changes in states due to the occurrence of certain events such as arrival of inputs. This provides a very general architecture for defining the dynamics of a system, except that, as indicated in the Introduction, it is inadequate for specifying complex real-time systems. Hierarchical Multi-State (HMS) machines [4] are high-level "multi-state automata," in which (1) multiple (hierarchical) states can be true at one moment, (2) multiple transitions can fire simultaneously, and (3) a temporal interval logic, called TIL, is used to define constraints on transitions. This architecture allows the compact definition of the dynamics of complex real-time systems, in which interactions among states and hard deadlines can be defined formally. In addition, a "multi-level" combination of HMS machines [5] provides the capability for formally defining dynamic *requirements*, giving rise to a model-based reasoning framework for real-time systems. Because of limitations of space, only the non-hierarchical version of HMS machines will be considered here. A formalization of hierarchies can be found in [6].

An HMS machine is a triple $H = (S, \Gamma_D, \Gamma_N)$, where S is a set of "states," Γ_D is a set of "deterministic" transitions, and Γ_N is a set of "nondeterministic" transitions. Boolean states represent properties that may be true or false about a system. Non-boolean states can represent both properties of multiple entities in a system and properties of data objects. Deterministic transitions denote *fixed* causal interactions among states, while nondeterministic transitions represent *possible* or *permissible* interactions. Nondeterminism, in fact, is the key to the specification of *choice* in model-based reasoning in the HMS framework.

The constraints or "controls" on transitions in an HMS machine are defined in terms of the temporal interval logic TIL which is obtained by adding the following three operators to propositional logic:

O(t): *At* relative time t

$[t_1, t_2]$: *Always* between times t_1 and t_2

$<t_1, t_2>$: *Sometime* between times t_1 and t_2

The operators $[t_1, t_2]$ and $<t_1, t_2>$, which allow hard real-time constraints to be defined for HMS machines, are generalizations of the standard temporal logic operators $\Box$ and $\Diamond$, respectively. All times are relative, with the current moment denoted by 0. Figure 2 depicts a simple 2-level example of an HMS machine specification that defines both a nondeterministic "basic machine" H_1 and a specification of requirements in terms of the "policy HMS

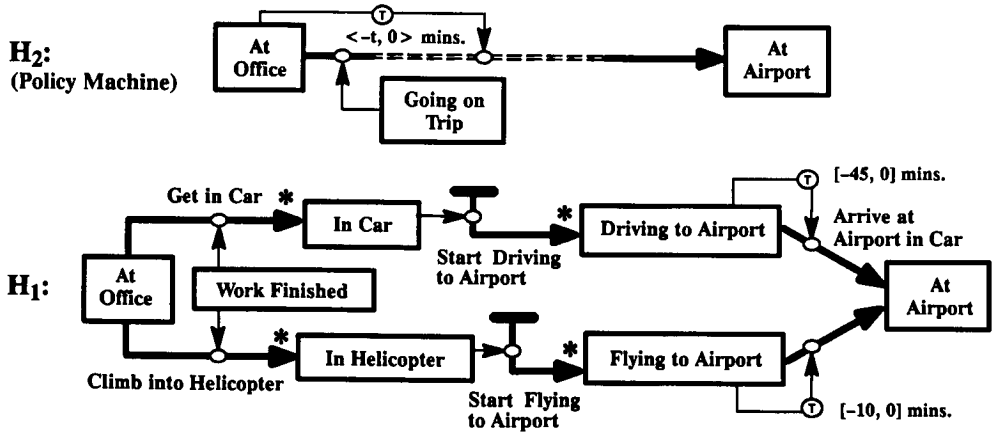

Figure 2. A 2-Level HMS Machine Specification of System and Requirements

machine" H_2. In this figure, rectangular boxes represent states, dark arrows are transitions, thin arrows denote TIL controls on transitions with the symbol ⓣ next to each temporal operator, and the partially double-dashed arrow in H_2 is a "policy transition" that defines intentionality. Asterisks denote nondeterministic transitions so that in H_1 the choice of all actions is not completely determined. We say that a transition is "enabled" if (1) its "primary" states from which the transition emerges are true, and (2) its controls are true. Thus, starting at the left side in the machine H_1, from the state "At Office" one can go into state "In Car" or into state "In Helicopter" as long as the control state "Work Finished" is true. If the state "In Car" ("In Helicopter") is true, then nondeterministically the transition "Start Driving to Airport" ("Start Flying to Airport") is fired. Nondeterminism is useful since this machine may be part of the specification of a much larger set of behaviors that could include going to many other destinations. The horizontal bar from which the transition "Start Driving to Airport" arises is an infinite resource which is always true. Thus, if this transition fires, both the states "In Car" and "Driving to Airport" would be true simultaneously. We note that at the end of this path, if the state "Driving to Airport" has been true continuously from 45 minutes earlier to the current moment, then a deterministic transition will take one to the state "At Airport."

The policy transition of machine H_2 in Figure 2 defines the goal of reaching the state "At Airport" when executing H_1, with the requirement that the state "Going on Trip" must be true in the beginning and the trip should not take more than t minutes. Thus, depending on the value of t, different "plans" for H_1 can be derived to reach the goal state. If the execution of the plan takes more than t minutes, then the plan can essentially be abandoned. Additional types of controls on policy transitions that are not shown in the figure can be used to define complex interactions of states and goals, including the capability of making a goal dependent on the planning process itself. Thus, for example, an alternate goal can be specified if the plan generation process takes longer than a specified length of time. Heuristic knowledge about plans can be captured by intermediate policy machines that define midpoint states that must be achieved during the execution of a plan. More details about policy machines can be found in [5].

An important benefit of the formal specification of a real-time system is that it provides a framework for verification of correctness and consistency before implementation. Following the procedure in 3, given an HMS machine and any safety property defined on its states, one can create a new "extended" state that will be true if and only if the safety property is violated. By a result of [8], such a state need only depend on the past history of the states of the machine, even though safety properties are usually defined in terms of future events. Two specific verification methods can then be used to verify that the extended state corresponding to the safety property is not reachable. In the first method, correctness-preserving transformations [3] are applied to modify an HMS machine incrementally, without affecting its behavior, until the safety state is isolated. In the second method, a "model-checking" approach [6] is used to demonstrate in finite time the correctness of infinite behavior. As in [2], this involves a branching simulation process that terminates paths when cycles are detected. A major advantage of using HMS machines is that orders of magnitude reduction in the number of states can be obtained in many applications compared to traditional automata models.

3. Planning, Plan Formalisms and Scheduling of Plans for HMS Machines

A "plan" in the HMS framework consists of a sequence of sets of transitions to be executed in a nondeterministic machine [5]. Conditional goals are specified for an HMS machine in terms of policy transitions of a policy HMS machine such as H_2 in Figure 2. The "planning" process then consists of searching the space of eligible nondeterministic transitions in a basic machine such as H_1 to derive a plan that causes the goal states of a policy transition

to be reached. The important points to note in this framework are that (1) goals can be defined formally in terms of histories of states that are being modified dynamically, (2) circumstances such as inability to meet a deadline may cause a goal to be dropped from consideration, (3) the states of the computational resources in which planning is being performed may be used as controls on the policy transitions that define goals, and (4) heuristic guidelines for deriving plans can be specified in terms of intermediate policy machines.

Compilation of plans in advance to meet goals with hard deadlines has been proposed by number of authors (see, e.g., [9]). Various representation schemes for plans have also been proposed. For example, in [1] a Petri net model is used to define conditional actions that depend on facts that are true about the environment. The HMS machine framework offers a powerful capability to define complex concurrent plans that depend not only on the current states of the world but also on temporal histories of states. For this purpose, we say that a machine P is a "plan HMS machine" for a nondeterministic machine H, if some of the states of P correspond to the nondeterministic transitions of H and some other states are "dependent" states of the states of H. A dependent state is defined as a state for which (1) truth only depends on a logical combination of the truth or falsehood of other states, and (2) there are certain restrictions on transitions emerging from it and entering it. At each moment of time, the "execution" of P on H then is obtained by (1) firing the transitions of P as in a standard HMS machine, (2) firing the deterministic transitions of H, and (3) firing those nondeterministic transitions of H that are enabled in H and for which a corresponding state in P is true. Thus, for example, the plan machine in Figure 3 describes how the nondeterministic transitions in the machine H_1 should be executed. The states containing asterisks are dependent states which, in this case, are simply duplicates of corresponding states in H_1, assuming that the state "In a Hurry" is added to H_1. The states denoted by dashed rectangles represent *transitions* in H_1. Thus, this machine indicates that in case the state "In a Hurry" is true, one should execute the transition "Climb into Helicopter" from the state "At Office" in H_1. On the other hand, if the state "In a Hurry" is false, the transition "Get in Car" should be executed. Also, when the state "In Car" becomes true in H_1, the transition "Start Driving to Airport" will be fired if its corresponding state in Figure 3 is true. The latter situation will be true if the state "Going on Trip" has been true sometime earlier.

Two simpler formalisms for defining HMS machine plans can be defined in terms of the plan languages PL_0 and PL_1, which can also be considered as languages for describing concurrent event sequences. Words in the language PL_0 simply consist of sequences of (1) symbols from the set of transitions of the HMS machine, (2) lists of symbols, (3) words with integer exponents. An individual symbol α denotes the firing of the corresponding transition in the machine. A list of the form $(\alpha, \beta, \ldots, \delta)$ denotes the simultaneous firings of the transitions $\alpha, \beta, \ldots, \delta$. A word of the form w^n represents the n–fold repetition of firing of the transitions in w. Thus, the plan $\alpha\ (\beta, \gamma)\ (\delta\eta)^n$ denotes the execution of the fol-

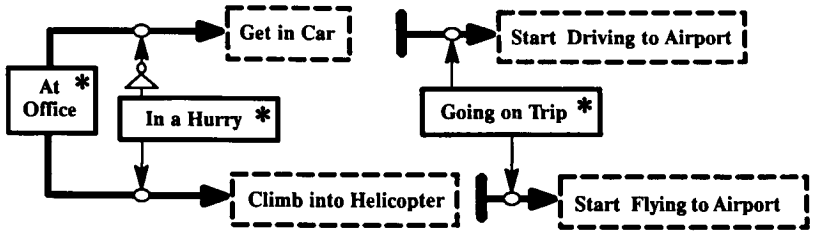

Figure 3. A Plan Machine for the HMS Machine H_1 in Figure 2.

lowing transitions in a machine: first fire α, then fire β and γ simultaneously, then fire δ followed by η n times. The language PL_1 extends PL_0 by the introduction of conditional operators and the means for defining alternative choices of actions. Plans in such languages, combined with an underlying HMS machine, provide the capability for both model-based reasoning from basic principles and the ability to respond rapidly to dynamic requirements without the need for searching.

Plan languages also offer the possibility of studying the *scheduling* of plans as distinct from the planning or plan generation process itself. For example, consider the plan

> "Get in Car" "Start Driving to Airport" "Arrive at Airport in Car"

in the plan language PL_0 for the machine H_1 of Figure 2. This plan simply lists the sequences of actions that must be performed, in which there is a key missing element: *when* should the actions be performed. Here, the only missing part is a delay of 45 minutes that must occur between the transitions "Start Driving to Airport" and "Arrive at Airport in Car." If such required delays are incorporated into a plan and it is verified for correctness, then the underlying machine can essentially be ignored during the execution. The important correctness criteria for plans are: (1) no transition is attempted that is not enabled, and (2) the plan will transform the machine from a given initial set of states to the desired final set of goal states.

In [5] a general approach for deriving schedules for plans was introduced that also provides a limited method of verifying the correctness of plans. In this scheme, given a potential plan p', a "variable delay plan" p is generated in which between each pair of terms in p' a parametric delay ϕ^i is introduced, where ϕ denotes a wait or "no action." Using symbolic execution techniques, then a solution for the exponents of the ϕ's can often be found that guarantees the correctness of the plan. In addition, in many cases, misordered plans can be corrected in the process of finding the delays.

4. Conclusions and Future Work

Hierarchical Multi-State (HMS) machines provide a framework for specification, verification and control of complex real-time systems by integrating multi-state automata and temporal interval logic. The major benefits are: (1) significant reduction in state space, (2) convenient mechanisms for specifying both safety properties and conditional goals, including hard deadlines, (3) methods of verifying correctness of specifications, and (4) model-based reasoning approaches for planning and scheduling in dynamic environments.

Three directions for future work have been defined: theory, applications and tools. Theoretical research goals include (1) the extension and formalization of the specification language, (2) investigation of more powerful methods for capturing requirements, (3) verification methods, (4) representation of uncertainty relating to both incomplete knowledge about the world and probabilistic outcome of events, (5) introduction of learning, and (6) efficient planning and scheduling algorithms. Various potential application areas for HMS machine have also been identified. Currently, HMS machines are being applied to the specification of a fragment of a future European command and control system. As far as tool development plans are concerned, work is continuing on the development of a prototype environment for specification of HMS machines, along with the capabilities for interactive simulation, limited forms of animation, and verification.

Acknowledgments

The author would like to acknowledge the contributions of Matthew K. Franklin in the development of the concepts discussed in this paper. Also, Ramachandran Iyer participated in the formalization of the temporal logic concepts discussed here and made helpful comments on an earlier draft.

References

[1] Drummond, M., "A representation of action and belief for automatic planning systems," *Proc. 1986 Workshop on Reasoning About Actions and Plans*, Morgan Kaufmann, 1987, pp. 189–212.

[2] Clarke, E.M., E.A. Emerson, A.P. Sistla, "Automatic verification of finite-state concurrent sytems using temporal logic," *ACM Transactions on Programming Languages and Systems*, Vol. 8, No. 2, 1986, pp. 244–263.

[3] Franklin, M.K., and A. Gabrielian, "A transformational method for verifying safety properties in real-time systems," *Proc. 10th Real-Time Systems Symposium*, Santa Monica, CA, Dec. 7–9, 1989, pp. 112–123.

[4] Gabrielian, A., and M.K. Franklin, "State-based specification of complex real-time systems," *Proc. 9th Real-Time Systems Symposium*, Huntsville, Dec. 10–12, 1988, pp. 2–11.

[5] Gabrielian, A., and M.K. Franklin, "Multi-level specification and verification of real-time software," *Proc. 12th International Conf. on Software Engineering*, Nice, France, March 26–30, 1990. To be reprinted in *Communications of the ACM*.

[6] Gabrielian, A. and R. Iyer, "Integrating automata and temporal logic: a framework for specification of real-time systems and software," *Proc. Unified Computation Laboratory*, Institute of Mathematics and its Applications, Stirling, Scotland, July 3–6, 1990, to appear.

[7] Gabrielian, A., and M.E. Stickney, "Hierarchical representation of causal knowledge," *Proceedings of WESTEX–87 IEEE Expert Systems Conference*, 1987, pp. 82–89.

[8] Manna, Z., and A. Pneuli, "The anchored version of the temporal framework," in *Proc. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science 354, Springer-Verlag, 1989, pp. 201–284.

[9] Schoppers, M.J., "Representation and automatic synthesis of reaction plans," *Ph.D. Thesis*, Computer Science Dept., Univ. of Illinois, Urbana-Champaign, 1989.

[10] Wolper, P., "Temporal logic can be more expressive," *Information and Control*, Vol. 1, No. 1–2, 1983, pp. 72–99.